

NATO STANDARDIZATION AGREEMENT
(STANAG)

SAFETY DESIGN REQUIREMENTS AND GUIDELINES FOR

- 5.3 The following functions are likely to be designated SCCSF's
- a. Any function which controls or directly influences the prearming, arming, enabling, release, launch, firing, or detonation of a munition system, including target identification, selection and designation.
 - b. Any function which determines, controls, or directly influences the flight path of a munition system.
 - c. Any function that controls or directly influences the movement of gun mounts, launchers, and other equipment, especially with respect to the pointing and firing safety of that equipment.
 - d. Any function which controls or directly influences the movement of munitions and/or hazardous materials.
 - e. Any function which monitors the state of the system for purposes of ensuring its safety.
 - f. Any function that senses hazards and/or displays information concerning the protection of the system.
 - g. Any function which controls or regulates energy sources in the system.
 - h. Fault detection priority. The priority structure of fault detection and restoration of safety or correcting logic shall be considered safety critical. Software units or modules handling or responding to these faults.
 - i. Interrupt processing software. Interrupt processing software, interrupt priority schemes and routines which disable or enable interrupts.
 - j. Autonomous control. Software components that have autonomous control over safety critical hardware.
 - k. Software controlled movement. Software that generates signals which have been shown through analysis to directly influence or control the movement of potentially hazardous hardware components or initiate safety critical actions.

1. Safety critical displays. Software that generates outputs that display the status of safety critical hardware systems. Where possible, these outputs shall be duplicated by non-software generated output.

- m. Critical data computation. Software used to compute safety critical data. This includes applications software that may not be connected to or directly control a safety critical hardware system (e.g., stress analysis programs).

Design and Development Process Requirements and Guidelines

6. The requirements of this section apply to the design and development phase.
 - 6.1 Configuration control. Configuration control shall be established as soon as practical in the system development process. All software changes occurring after an initial baseline has been established must be approved by the Software Configuration Control Board prior to their implementation. A member of the Board shall be tasked with the responsibility for evaluation of all software changes for their potential safety impact. This member should be a member of the system safety engineering team. A member of the hardware Configuration Control Board shall be a member of the Software Configuration Control Board and vice versa to keep members apprised of hardware changes and to ensure that software changes do not conflict with or introduce potential safety hazards due to hardware incompatibilities.
 - 6.2 Quality Assurance Program. A quality assurance program shall be established for systems having safety critical functions.
 - 6.3 Two person rule. At least two people shall be thoroughly familiar with the design, code, testing and operation of each software module in the system.
 - 6.4 Reviews and audits. Desk audits, peer reviews, static and dynamic analysis tools and techniques, and debugging tools shall be used to verify implementation of design requirements in the source code with particular attention paid to the implementation of identified safety critical computing system functions and the requirements and guidelines provided in this document. Reviews of the software source code shall ensure that the code and comments within the code agree.
 - 6.5 Program patch prohibition. Patches shall be prohibited throughout the development process. All software changes shall be coded in the source language and compiled prior to entry into operational or test equipment.
 - 6.6 Design verification/validation. The software shall be analyzed throughout the design, development, and maintenance process by a system safety engineering team to verify and validate that the safety design requirements have been correctly and completely implemented. Test results shall be analyzed to identify potential safety anomalies that may occur.

System Design Requirements and Guidelines

7. The requirements and guidelines of this section apply to the general system design.
 - 7.1 Designed safe states. The system shall have at least one safe state identified for each logistic and operational phase.
 - 7.2 Standalone Computer. Where practical, safety critical functions should be performed on a standalone computer. If this is not practical, safety critical functions shall be isolated to the maximum extent practical from non-critical functions.
 - 7.3 Ease of maintenance. The system and its software shall be designed for ease of maintenance by future personnel not associated with the original design team. Documentation specified for the computing system shall be developed to facilitate maintenance of the software. Strict configuration control of the software during development and after deployment is required. The use of techniques for the decomposition of the software system for ease of maintenance is recommended.
 - 7.4 Safe state return. The software shall return hardware subsystems under the control of software to a designed safe state when unsafe conditions are detected. Conditions that can be safely overridden by the battle short shall be identified and analyses performed to verify their safe incorporation.
 - 7.5 Restoration of interlocks. Upon completion of tests and/or training wherein safety interlocks are removed, disabled or bypassed, restoration of those interlocks shall be verified by the software prior to being able to resume normal operation. While overridden, a display shall be made on the operator's or test conductor's console of the status of the interlocks, if applicable.
 - 7.6 Input/output registers. Input/output registers and ports shall not be used for both safety critical and non-critical functions unless the same safety design criteria are applied to the non-critical functions.
 - 7.7 External hardware failures. The software shall be designed to detect failures in external hardware input or output hardware devices and revert to a safe state upon their occurrence. The design shall consider potential failure modes of the hardware involved.
 - 7.8 Safety kernel failure. The system shall be designed such that a failure of the safety kernel (when implemented) will be detected and the system returned to a designed safe state.
 - 7.9 Circumvent unsafe conditions. The system design shall not permit detected unsafe conditions to be circumvented. If a "battleshort" or "safety arc" condition is required in the system, it shall be designed such that it cannot be activated either inadvertently or without authorization.
 - 7.10 Fallback and recovery. The system shall be designed to include fallback and recovery to a designed safe state of reduced system functional capability in the event of a failure of system components.

- 7.11 Simulators. If simulated items, simulators, and test sets are required, the system shall be designed such that the identification of the devices is fail safe and that operational hardware cannot be inadvertently identified as a simulated item, simulator or test set.
- 7.12 System errors log. The software shall make provisions for logging all system errors detected. The operator shall have the capability to review logged system errors. Errors in safety critical routines shall be highlighted and shall be brought to the operator's attention as soon as practical after their occurrence.
- 7.13 Positive feedback mechanisms. Software control of critical functions shall have feedback mechanisms that give positive indications of the function's occurrence.
- 7.14 Peak load conditions. The system and software shall be designed to ensure that design safety requirements are not violated under peak load conditions.

Power-Up System Initialization Requirements

8. The following requirements apply to the design of the power subsystem, power control, and power-on initialization for safety critical applications of computing systems.

- 8.1 Power-up initialization. The system shall be designed to power-up in a safe state. An initialization test shall be incorporated in the design that verifies that the system is in a safe state and that safety critical circuits and components are tested to ensure their safe operation. The test shall also verify memory integrity and program load.
- 8.2 Power faults. The system and computing system shall be designed to ensure that the system is in a safe state during power-up, intermittent faults or fluctuations in the power that could adversely affect the system, or in the event of power loss. The system and/or software shall be designed to provide for a safe, orderly shutdown of the system due to either a fault or power-down, such that potentially unsafe states are not created.
- 8.3 Primary computer failure. The system shall be designed such that a failure of the primary control computer will be detected and the system returned to a safe state.
- 8.4 Maintenance interlocks. Maintenance interlocks, safety interlocks, safing handles, and/or safing pins shall be provided to preclude hazards to personnel maintaining the computing system and its associated equipment. Where interlocks, etc. must be overridden to perform tests or maintenance, they shall be designed such that they cannot be inadvertently overridden, or left in the overridden state once the system is restored to operational use. The override of the interlocks shall not be controlled by a computing system.
- 8.5 System level check. The software shall be designed to perform a system level check at power-up to verify that the system is safe and functioning properly prior to application of power to safety critical functions including hardware controlled by the software. Periodic tests shall be performed by the software to monitor the safe state of the system.

Computing System Hardware Requirements and Guidelines

9. The requirements and guidelines of this section apply to the design and selection of computers, microprocessors, programming languages, and memories for safety critical applications in munition systems.

9.1 CPU selection. The following guidelines apply to the selection of Central Processing Units (CPUs):

- a. CPUs that process entire instructions or data words are preferred to those that multiplex instructions or data (e.g., an 8-bit CPU is preferred to a 4-bit CPU emulating an 8-bit machine).
- b. CPUs with separate instruction and data memories and busses are preferred to those using a common data/instruction buss. Alternatively, memory protection hardware, either segment or page protection, separating program memory and data memory is acceptable.
- c. CPUs, microprocessors and computers that can be fully represented mathematically are preferred to those that cannot.

9.2 Minimum clock cycles. For CPUs that do not comply with the guidelines of 9.1, or those used at the limits of their design criteria (e.g., at or above maximum clock frequency), analyses and measurements shall be conducted to determine the minimum number of clock cycles that must occur between functions on the buss to ensure that invalid information is not picked up by the CPU. Analyses shall also be performed to ensure that interfacing devices are capable of providing valid data within the required time frame for CPU access.

9.3 Read-only-memories. Where Read-Only-Memories are used, positive measures shall be taken to ensure that the data cannot be corrupted or destroyed.

Self-Checking Design Requirements and Guidelines

10. The design requirements of this section provide for self checking of the programs and computing system execution.

- 10.1 Watchdog timers. Watchdog timers or similar devices shall be provided to ensure that the microprocessor or computer is operating properly. The timer reset shall be designed such that the software cannot enter an inner loop and reset the timer as part of that loop sequence. The design of the timer shall ensure that failure of the primary CPU clock cannot compromise its function. The timer reset shall be designed such that the system is returned to a known safe state and the operator alerted (as applicable).
- 10.2 Memory checks. Periodic checks of memory, instruction, and data buss(es) shall be performed. The design of the test sequence shall ensure that single point or likely multiple failures are detected. Checksum of data transfers and Program Load Verification (PLV) checks shall be performed at load time and periodically thereafter to ensure the integrity of safety critical code.
- 10.3 Fault detection. Fault detection and isolation programs shall be written for safety critical subsystems of the computing system. The fault detection program shall be designed to detect potential safety critical failures prior to execution of the related safety critical function. Fault isolation programs shall be designed to isolate the fault to the lowest level practical and provide this information to the operator or maintainer.
- 10.4 Operational checks. Operational checks of testable safety critical system elements shall be made immediately prior to performance of a related safety critical operation.

Safety Critical Computing System Function Protection Requirements and Guidelines

11. The design requirements and guidelines of this section provide for protection of safety critical computing system functions and data.

- 11.1 Safety degradation. The system shall be designed such that automata and software shall prevent degradation of safety by other interfacing automata and software.
- 11.2 Unauthorized interaction. The software shall be designed to prevent unauthorized system or subsystem interaction from initiating or sustaining a safety critical function sequence.
- 11.3 Unauthorized access. The system design shall prevent unauthorized or inadvertent access to or modification of the software (source or assembly) and object code. This includes preventing self-modification of the code.
- 11.4 Safety kernel ROM. Safety kernels should be resident in non-volatile read only memory (ROM) or in protected memory that cannot be overwritten by the computing system.
- 11.5 Safety kernel independence. A safety kernel, if implemented, shall be designed and implemented in such a manner that it cannot be corrupted, misdirected, delayed or inhibited by any other program in the system.
- 11.6 Inadvertent jumps. The system shall detect inadvertent jumps within or into Safety Critical Computing System Functions, return the system to a safe state, and, if practical, perform diagnostics and fault isolation to determine the cause of the inadvertent jump.
- 11.7 Load data integrity. The executive program or operating system shall ensure the integrity of data or programs loaded into memory prior to their execution.
- 11.8 Operational reconfiguration integrity. The executive program or operating system shall ensure the integrity of the data and programs during operational reconfiguration.

Interface Design Requirements

12. The design requirements of this section apply to the design of input/output interfaces.
 - 12.1 Feedback loops. Feedback loops from the system hardware shall be designed such that the software cannot cause a runaway condition due to the failure of a feedback sensor. Known component failure modes shall be considered in the design of the software and checks designed into the software to detect failures.
 - 12.2 Interface control. Safety Critical Computing System Functions and their interfaces to safety critical hardware shall be controlled at all times, i.e., the interface shall be monitored to ensure that erroneous or spurious data does not adversely affect the system, that interface failures are detected, and that the state of the interface is safe during power-up, power fluctuations & interruptions, or in the event of system errors or hardware failures.
 - 12.3 Decision statements. Decision statements in safety critical computing system functions shall not rely on inputs of all ones or all zeros, particularly when this information is obtained from external sensors.
 - 12.4 Inter-CPU communications. Inter-CPU communications shall successfully pass verification checks in both CPUs prior to the transfer of safety critical data. Periodic checks shall be performed to ensure the integrity of the interface. Detected errors shall be logged. If the interface fails several consecutive transfers, the operator shall be alerted and the transfer of safety critical data terminated until diagnostic checks can be performed.
 - 12.5 Data transfer messages. Data transfer messages shall be of a predetermined format and content. Each transfer shall contain a word or character string indicating the message length (if variable), the type of data and content of the message. As a minimum, parity checks and checksums shall be used for verification of correct data transfer. Cyclic Redundancy Checks (CRCs) shall be used where practical. No information from data transfer messages shall be used prior to verification of correct data transfer.
 - 12.6 External functions. External functions requiring two or more safety critical signals from the software (e.g., arming of an Ignition Safety Device or Arm Fire Device and release of an air launched weapon) shall not receive all of the necessary signals from a single input/output register or buffer.
 - 12.7 Input reasonableness checks. Limit and reasonableness checks, including time limits, dependencies, and reasonableness checks, shall be performed on all analog and digital inputs and outputs prior to safety critical functions' execution based on those values. No safety critical functions shall be executable based on safety critical analog or digital inputs that cannot be verified.
 - 12.8 Full scale representations. The software shall be designed such that the full scale and zero representations of the software are fully compatible with the scales of any digital-to-analog, analog-to-digital, digital-to-synchro, and/or synchro-to-digital converters.

13. The design requirements of this section apply to the design of the user interface to safety critical computing systems.

- 13.1 Processing cancellation. The software shall be designed such that the operator may cancel current processing with a single action and have the system revert to a designed safe state. The system shall be designed such that the operator may exit potentially unsafe states with a single action. This action shall revert the system to a known safe state. (e.g., the operator shall be able to terminate missile launch processing with a single action which shall safe the missile.) The action may consist of pressing two keys, buttons, or switches at the same time. Where operator reaction time is not sufficient to prevent a mishap, the software shall revert the system to a known safe state, report the failure, and report the system status to the operator.
- 13.2 Hazardous function initiation. Two or more unique operator actions shall be required to initiate any potentially hazardous function or sequence of functions. The actions required shall be designed to minimize the potential for inadvertent actuation, and shall be checked for proper sequence.
- 13.3 Safety critical displays. Safety critical operator displays, legends and other interface functions shall be clear, concise and unambiguous and, where possible, be duplicated using separate display devices.
- 13.4 Operator entry errors. The software shall be capable of detecting improper operator entries or sequences of entries or operations and prevent execution of safety critical functions as a result. It shall alert the operator to the erroneous entry or operation. Alerts shall indicate the error and corrective action. The software shall also provide positive confirmation of valid data entry or actions taken (i.e., the system shall provide visual and/or aural feedback to the operator such that the operator knows that the system has accepted the action and is processing it). The system shall also provide a real-time indication that it is functioning. Processing functions requiring several seconds or longer shall provide a status indicator to the operator during processing.
- 13.5 Safety critical alerts. Alerts shall be designed such that routine alerts are readily distinguished from safety critical alerts. The operator shall not be able to clear a safety critical alert without taking corrective action or performing subsequent actions required to complete the ongoing operation.
- 13.6 Unsafe situation alerts. Signals alerting the operator to unsafe situations shall be directed as straightforward as practical to the operator interface.
- 13.7 Unsafe state alerts. If an operator interface is provided and a potentially unsafe state has been detected, the system shall alert the operator to the anomaly detected, the action taken, and the resulting system configuration and status.

Software Design and Coding Requirements and Guidelines

14. The requirements and guidelines of this section apply to the design of the software.
 - 14.1 Modular code. Software design and code shall be modular. Modules shall have one entry and one exit point.
 - 14.2 Number of modules. The number of program modules containing safety critical functions shall be minimized where possible within the constraints of operational effectiveness, computer resources and good software design practices.
 - 14.3 Execution path. Safety Critical Computing System Functions shall have one and only one possible path leading to their execution.
 - 14.4 Halt instructions. Halt, stop or wait instructions shall not be used in code for safety critical functions. Wait instructions may be used where necessary to synchronize Input/ Output, etc. and when appropriate handshake signals are not available. See also requirements 14.10 and 14.11.
 - 14.5 Single purpose files. Files used to store safety critical data shall be unique and shall have a single purpose. Scratch files, those used for temporary storage of data during or between processes, shall not be used for storing or transferring safety critical information, data, or control functions.
 - 14.6 Unnecessary features. The operational and support software shall contain only those features and capabilities required by the system. The programs shall not contain undocumented or unnecessary features.
 - 14.7 Indirect addressing methods. Indirect addressing methods shall be used only in well controlled applications. When used, the address shall be verified as being within acceptable limits prior to execution of safety critical operations. Data written to arrays in safety critical applications shall have the address boundary checked by the compiled code.
 - 14.8 Uninterruptable code. If interrupts are used, sections of the code which have been defined as uninterruptable shall have defined execution times monitored by an external timer.
 - 14.9 Safety critical files. Files used to store or transfer safety critical information shall be initialized to a known state before and after use. Data transfers and data stores shall be audited where practical to allow traceability of system functioning.
 - 14.10 Unused memory. All processor memory not used for or by the operational program shall be initialized to a pattern that will cause the system to revert to a safe state if executed. It shall not be filled with random numbers, halt, stop, wait or no-operation instructions. Data or code from previous overlays or loads shall not be allowed to remain. (Examples: If the processor architecture halts upon receipt of non-executable code, a watchdog timer shall be provided with an interrupt routine to revert the system to a safe state. If the processor flags non-executable code as an error, an error handling routine shall be developed to revert the system to a safe state and

terminate processing.) Information shall be provided to the operator to alert him to the failure or fault observed and to inform him of the resultant safe state to which the system was reverted.

- 14.11 Overlays. Overlays of safety critical software shall all occupy the same amount of memory. Where less memory is required for a particular function, the remainder shall be filled with a pattern that will cause the system to revert to a safe state if executed. It shall not be filled with random numbers, halt, stop, no-op or wait instructions or data or code from previous overlays.
- 14.12 Operating system functions. If an operating system function is provided to accomplish a specific task, operational programs shall use that function and not bypass it or implement it in another fashion.

15. The following design requirements and guidelines apply to the software coding phase.

- 15.1 Compilers. The implementation of software compilers shall be validated to ensure that the compiled code is fully compatible with the target computing system and application (may be done once for a target computing system).
- 15.2 Flags and variables. Flags and variable names shall be unique. Flags and variables shall have a single purpose and shall be defined and initialized prior to use.
- 15.3 Loop entry point. Loops shall have one and only one entry point. Branches into loops shall not be used. Branches out of loops shall lead to a single exit point placed after the loop within the same module.
- 15.4 Software maintenance design. The software shall be annotated, designed, and documented for ease of analysis, maintenance, and testing of future changes to the software.
- 15.5 Variable declaration. Variables or constants used by a safety critical function will be declared/initialized at the lowest possible level
- 15.6 Unused executable code. Operational program loads shall not contain unused executable code.
- 15.7 Unreferenced variables. Operational program loads shall not contain unreferenced or unused variables or constants.
- 15.8 Assignment statements. Safety Critical Computing System Functions and other safety critical software items shall not be used in one-to-one assignment statements unless the other variable is also designated as safety critical (e.g., shall not be redefined as another non-safety critical variable).
- 15.9 Conditional statements. Conditional statements shall have all possible conditions satisfied and under full software control (i.e. there shall be no potential unresolved input to the conditional statement). Conditional statements shall be analyzed to ensure that the conditions are reasonable for the task and that all potential conditions are satisfied and not left to a default condition. All condition statements shall be annotated with their purpose and expected outcome for given conditions.
- 15.10 Strong data typing. Safety critical functions shall exhibit strong data typing. Safety critical functions shall not employ a logic "1" and "0" to denote the safe and armed (potentially hazardous) states. The armed and safe states for munitions shall be represented by at least a unique four bit pattern. The safe state shall be a pattern that cannot, as a result of a one, two, or three bit error, represent the armed pattern. The armed pattern shall also not be the inverse of the safe pattern. If a pattern other than these two unique codes is detected, the software shall flag the error, revert to a safe state and notify the operator, if appropriate.

- 15.11 Timer values annotated. Values for timers shall be annotated in the code. Comments shall include a description of the timer function, its value and the rationale or a reference to the documentation explaining the rationale for the timer value. These values shall be verified and shall be examined for reasonableness for the intended function.
- 15.12 Critical variable identification. Safety critical variables shall be identified in such a manner that they can be readily distinguished from non-safety critical variables (e.g., all safety critical variables begin with a letter S).
- 15.13 Global Variables. Global variables shall not be used for safety critical functions.

Software Maintenance Requirements and Guidelines

16. The requirements and guidelines of this section are applicable to the maintenance of the software in safety critical computing system applications. The requirements applicable to the design and development phase as well as the software design and coding phase are also applicable to the maintenance of the computing system and software.

- 16.1 Critical function changes. Changes to safety critical computing system functions on deployed or fielded systems shall be issued as a complete package for the modified unit or module and shall not be patched.
- 16.2 Critical firmware changes. When not implemented at the depot level or in manufacturers facilities under appropriate quality control, firmware changes shall be issued as a fully functional and tested circuit card. Design of the card and the installation procedures should minimize the potential for damage to the circuits due to mishandling, electrostatic discharge, or normal or abnormal storage environments, and shall be accompanied with the proper installation procedure.
- 16.3 Software change medium. When not implemented at the depot level or in manufacturers facilities under appropriate quality control, software changes shall be issued as a fully functional copy on the appropriate medium. The medium, its packaging and the procedures for loading the program should minimize the potential damage to the medium due to mishandling, electrostatic discharge, potential magnetic fields, or normal or abnormal storage environments, and shall be accompanied with the proper installation procedure.
- 16.4 Modification configuration control. All modifications and updates shall be subject to strict configuration control. The use of automated configuration management tools is encouraged.
- 16.5 Version identification. Modified software or firmware shall be clearly identified with the version of the modification, including configuration control information. Both physical (e.g., external label) and electronic (i.e., internal digital identification) "fingerprinting" of the version shall be used.

Software Analysis and Testing

17. The requirements of this section are applicable to the software testing phase.
 - 17.1 Formal test coverage. All software testing shall be controlled by a formal test coverage analysis and document. Computer based tools shall be used to ensure that the coverage is as complete as possible.
 - 17.2 Go/No-Go path testing. Software testing shall include GO-NO-GO path testing.
 - 17.3 Input failure modes. Software testing shall include hardware and software input failure mode testing.
 - 17.4 Boundary test conditions. Software testing shall include boundary, out-of-bounds and boundary crossing test conditions.
 - 17.5 Input data rates. Software testing shall include minimum and maximum input data rates in worst case configurations to determine the system's capabilities and responses to these conditions.
 - 17.6 Zero value testing. Software testing shall include input values of zero, zero crossing and approaching zero from either direction and similar values for trigonometric functions.
 - 17.7 Regression testing. Safety critical computing system functions in which changes have been made shall be subjected to complete regression testing.
 - 17.8 Operator interface testing. Operator interface testing shall include operator errors during safety critical operations to verify safe system response to these errors.
 - 17.9 Duration stress testing. Software testing shall include duration stress testing. The stress test time shall be continued for at least the maximum expected operating time for the system. Testing shall be conducted under simulated operational environments. Additional stress duration testing should be conducted to identify potential critical functions (e.g., timing, data senescence, resource exhaustion, etc.) that are adversely affected as a result of operational duration. Software testing shall include throughput stress testing (e.g., CPU, data bus, memory, input/output) under peak loading conditions.

Critical Timing and Interrupt Functions

18. The following design requirements and guidelines apply to safety critical timing functions and interrupts.

- 18.1 Safety critical timing. Safety critical timing functions shall be controlled by the computer and shall not rely on human input. Safety critical timing values shall not be modifiable by the operator from system consoles, unless specifically required by the system design. In these instances, the computer shall determine the reasonableness of timing values.
- 18.2 Valid interrupts. The software shall be capable of discriminating between valid and invalid (e.g. spurious) external and/or internal interrupts. Invalid interrupts shall not be capable of creating hazardous conditions. Valid external and internal interrupts shall be defined in system specifications. Internal software interrupts are not a preferred design as they reduce the analyzability of the system.
- 18.3 Recursive loops. Recursive and iterative loops shall have a maximum documented execution time. Reasonableness checks will be performed to prevent loops from exceeding the maximum execution time.
- 18.4 Time dependency. The results of a program should not be dependent on the time taken to execute the program or the time at which execution is initiated. Safety critical routines in real-time programs shall ensure that the data used is still valid (e.g., by using senescence checks).

IMPLEMENTATION OF THE AGREEMENT

19. The STANAG is implemented when NATO Members have adopted these requirements and guidelines for use in the design of safety critical applications of computing systems.

Applicable NATO Documents

Standardization Agreements (STANAGs)

- 1307 Maximum NATO Naval Operational Electromagnetic Environment Produced by Radio and Radar
- Draft
2818
(Ed. 2) Demolition Stores, Principles for Safe Design
- 3094 AVS, 16 Bit Architecture for Avionics Applications
- 3441 Design of Aircraft Stores for Fixed Wing Aircraft and Helicopters.
- 3838 PVS, Digital Time Division Command /Response Multiplex Data Bus
- 3912 AVS, Standardization of Real Time High Order Computer Programming Language for Avionics Applications
- 4145 Nuclear Survivability Criteria for Armed Forces, Materials, and Installations. AEP-4 (C)
- 4187 Fuzing Systems - Safety Design Requirements
- 4224 Large Calibre Artillery and Naval Gun Ammunition Greater Than
(Ed. 2) 40mm, Safety and Suitability for Service Evaluation
- 4225 Field Mortar Munitions, Safety Evaluation
(Ed. 2)
- 4228 Gun Munitions of Caliber 20 to 40 mm, Safety Evaluation
- 4234 Electromagnetic Radiation (Radio Frequency) 200 KHz to 40 GHz Environment - Affecting the design of Materiel for Use by NATO Forces
- 4235 Electrostatic Environmental Conditions Affecting the Design of Materiel for Use by NATO Forces
- 4236 Lightning Environmental Conditions Affecting the Design of Materiel for Use by NATO Forces
- 4239 Electrostatic Discharge Test Procedures to Determine the Safety and Suitability for Service of EED's and Associated Electronic Systems in Munitions and Weapon Systems
- 4297 Guidance on the Assessment of the Safety and Suitability for Service of Munitions for NATO Armed Forces AOP-15.

NATO UNCLASSIFIED

Annex A to
STANAG 4404
(Edition 1)

- 4324 Electromagnetic Radiation (Radio Frequency) Test Information to Determine the Safety and Suitability for Service of Electro-Explosive Devices and Associated Electronic Systems in Munitions and Weapon Systems
- 4325 Air-Launched Munitions, Safety Evaluation
- 4327 Lightning Test Procedures to Determine the Safety and Suitability for Service of EED's and Associated Electronic Systems in Munitions and Weapon Systems
- 4333 Underwater Munitions, Principles for Safe Design
- 4337 Surface Launched Munitions, Safety Evaluation
- 4338 Underwater-Launched Munitions, Safety Evaluation
- 4370 NATO environmental conditions and test procedures
- 4423 Aircraft Guns & Ammunition, Safety Evaluation
- 4432 Air Launched Guided Munitions, Principles for Safe Design
- 4433 Field Mortar Munitions, Principles for Safe Design
- (Draft)
4452 Safety Assessment Requirements for Munition Related Computing Systems

Other Publications:

- AOP-15 Guidance on the Assessment of the Safety and Suitability for Service of Munitions for NATO Armed Forces
- AQAP-150 NATO Software Quality Control System Requirements.
- AQAP-153 Guide for the Evaluation of a Contractor's Software Quality Control System for Compliance with AQAP-13.

Tailoring Guideline

Table B-1 provides a guideline for tailoring the requirements and guidelines contained in the body of this STANAG to different type, size, or complexity of systems. The following notations are used in tables B-1 and B-4:

R	Recommend implementation of design requirement or guideline in this system category or during this development phase
O	Optional design requirement or guideline to this system category or development phase
N	Not applicable in general to this system category or development phase
V	Verify implementation during this development phase (applicable to Table B-4 only).
T	Test implementation during this development phase (applicable to Table B-4 only).

The size, type and complexity of munition systems are broken down into the following six categories:

F&E	Fuzes and Electronic Safety and Arming Devices
S	Seekers
GS	Guidance Systems
T&L	Targeting and Launcher Control Systems
P&F	Pointing and Firing Cutout Systems
C&C	Command and Control Systems

The guideline is further broken down according to the criticality of the application of the computing system to the munition under consideration using the software control categories listed in table B-2. Items falling into control category I are considered highly critical, category II as critical and category III as marginal. Category IV is considered non critical. For example, "intelligent" fuzes and Safety and Arming Devices with separate mechanical safe and arm devices are considered critical whereas "intelligent" fuzes with computer controlled safe and arm devices are considered highly critical. Table B-3 provides a breakdown of the various munition system examples and their control categories.

Table B-4 provides additional guidance as to the program phase in which a particular guideline is best applied. The program phases are:

C	Conceptual design and feasibility studies phase
PD	Preliminary or early design phase
DD	Detailed design phase
Code	Software coding or component building phase
Test	Unit or module testing
Integ	System and software integration testing
Mods	Software enhancements, modifications, or maintenance

The same letters used in table B-1 indicate the level of applicability of the guideline to the particular program development phase. Recommended requirements and guidelines are those that are generally most effective if implemented during the indicated program development phase. Testing and verification of specific guidelines accomplished during the modifications phase should be in accordance with the recommendation for the unit and integration test recommendations.

Table B-1
Design Requirement and Guideline Tailoring
System Type/Guideline Applicability

Design Guideline Reference	F&E	S	GS	T&L	P&F	C&C
6. Design and Development Process						
6.1 Configuration control	R	R	R	R	R	R
6.2 Quality Assurance Program	R	R	R	R	R	R
6.3 Two person rule	R	R	R	R	R	R
6.4 Reviews and audits	R	R	R	R	R	R
6.5 Program patch prohibition	R	R	R	R	R	R
6.6 Design Verification/Validation	R	R	R	R	R	R
7. System Design Requirements						
7.1 Designed safe states.	R	R	R	R	R	R
7.2 Standalone computer	O	O	O	O	O	O
7.3 Ease of maintenance	R	R	R	R	R	R
7.4 Safe state return	N	N	O	R	R	R
7.5 Restoration of interlocks	N	N	N	R	R	R
7.6 Input/Output registers	O	O	R	R	R	R
7.7 External hardware failures	O	O	O	R	R	R
7.8 Safety kernel failure	R	R	R	R	R	R
7.9 Circumvent unsafe conditions	N	N	N	R	R	R
7.10 Fallback and recovery	N	N	N	R	R	R
7.11 Simulators	R	R	R	R	R	R
7.12 System error log	N	N	N	R	R	R
7.13 Positive feedback mechanisms	R	R	R	R	R	R
7.14 Peak load conditions	R	R	R	R	R	R

Annex B to
STANAG 4404
(Edition 1)

Table B-1 (Continued)
Design Requirement and Guideline Tailoring
System Type/Guideline Applicability

Design Guideline Reference	F&E	S	GS	T&L	P&F	C&C
8. Power-up Initialization Requirements						
8.1 Power Up initialization	R	R	R	R	R	R
8.2 Power faults	R	R	R	R	R	R
8.3 Primary computer failure	R	R	R	R	R	R
8.4 Maintenance interlocks	N	N	N	R	R	R
8.5 System level check	R	R	R	R	R	R
9. Computing System Hardware Requirements and Guidelines						
9.1 CPU Selection	R	R	R	R	R	R
9.2 Minimum clock cycles	R	R	R	R	R	R
9.3 Read only memories	R	R	R	R	R	R
10. Self-Check Design Requirements and Guidelines						
10.1 Watchdog timers	R	R	R	R	R	R
10.2 Memory checks	O	O	O	R	R	R
10.3 Fault detection	N	N	N	O	R	R
10.4 Operational checks	R	R	R	R	R	R
11. Safety Critical Computing System Functions Protection						
11.1 Safety degradation	R	R	R	R	R	R
11.2 Unauthorized interaction	N	N	R	O	R	R
11.3 Unauthorized access	R	R	R	R	R	R
11.4 Safety kernels ROM	R	R	R	R	R	R
11.5 Safety kernel independence	O	O	R	R	R	R
11.6 Inadvertent jumps	R	R	R	R	R	R
11.7 Load data integrity	R	R	R	R	R	R
11.8 Operational reconfiguration integrity	N	N	N	O	R	R

Table B-1 (Continued)
Design Requirement and Guideline Tailoring
System Type/Guideline Applicability

Design Guideline Reference	F&E	S	GS	T&L	P&F	C&C
12. Interface Design Requirements						
12.1 Feedback loops	R	R	R	R	R	R
12.2 Interface control	R	R	R	R	R	R
12.3 Decision statements	R	R	R	R	R	R
12.4 Inter-CPU communications	R	R	R	R	R	R
12.5 Data transfer messages	R	R	R	R	R	R
12.6 External functions	O	O	R	R	R	R
12.7 Input reasonableness checks	O	O	R	R	R	R
12.8 Full scale representations	R	R	R	R	R	R
13. User Interface Design						
13.1 Processing cancellation	N	N	N	R	R	R
13.2 Hazardous function initiation	N	N	N	R	R	R
13.3 Safety critical displays	N	N	N	R	R	R
13.4 Operator entry errors	N	N	N	R	R	R
13.5 Safety critical alerts	N	N	N	R	R	R
13.6 Unsafe situation alerts	N	N	N	R	R	R
13.7 Unsafe state alerts	R	R	R	R	R	R
14. Software Design and Coding Requirements						
14.1 Modular code	R	R	R	R	R	R
14.2 Number of modules	R	R	R	R	R	R
14.3 Execution path	R	R	R	R	R	R
14.4 Halt instructions	R	R	R	R	R	R
14.5 Single purpose files	N	R	R	R	R	R
14.6 Unnecessary features	R	R	R	R	R	R

Annex B to
STANAG 4404
(Edition 1)

Table B-1 (Continued)
Design Requirement and Guideline Tailoring
System Type/Guideline Applicability

Design Guideline Reference	F&E	S	GS	T&L	P&F	C&C
14. Software Design (continued)						
14.7 Indirect addressing methods	O	R	R	R	R	R
14.8 Uninterruptable code	R	R	R	R	R	R
14.9 Safety critical files	R	R	R	R	R	R
14.10 Unused memory	R	R	R	R	R	R
14.11 Overlays	N	N	O	O	R	R
14.12 Operating system functions	N	N	O	R	R	R
15. Software Coding Requirements						
15.1 Compilers	R	R	R	R	R	R
15.2 Flags and variables	R	R	R	R	R	R
15.3 Loop entry point	R	R	R	R	R	R
15.4 Software maintenance design	R	R	R	R	R	R
15.5 Variable declaration	R	R	R	R	R	R
15.6 Unused executable code	R	R	R	R	R	R
15.7 Unreferenced variables	R	R	R	R	R	R
15.8 Assignment statements	O	O	O	R	R	R
15.9 Conditional statements	R	R	R	R	R	R
15.10 Strong data typing	R	R	R	R	R	R
15.11 Timer values annotated	R	R	R	R	R	R
15.12 Critical variable identification	R	R	R	R	R	R
15.13 Global Variables	R	R	R	R	R	R

Table B-1 (Continued)
Design Requirement and Guideline Tailoring
System Type/Guideline Applicability

Design Guideline Reference	F&E	S	GS	T&L	P&F	C&C
16. Software Maintenance						
16.1 Critical function changes	R	R	R	R	R	R
16.2 Critical firmware changes	R	R	R	R	R	R
16.3 Software change medium	N	N	N	R	R	R
16.4 Modification configuration control	R	R	R	R	R	R
16.5 Version identification	R	R	R	R	R	R
17. Software Analysis and Testing						
17.1 Formal test coverage	R	R	R	R	R	R
17.2 GO/NO-GO path testing	R	R	R	R	R	R
17.3 Input failure modes	R	R	R	R	R	R
17.4 Boundary test conditions	O	R	R	R	R	R
17.5 Input data rates	O	R	R	R	R	R
17.6 Zero value testing	O	R	R	R	R	R
17.7 Regression testing	R	R	R	R	R	R
17.8 Operator interface testing	N	N	N	R	R	R
17.9 Duration stress testing	R	R	R	R	R	R
18. Critical Timing and Interrupt Functions						
18.1 Safety critical timing	N	N	O	O	R	R
18.2 Valid interrupts	O	O	O	R	R	R
18.3 Recursive loops	N	N	R	R	R	R
18.4 Time dependency	R	R	R	R	R	R

Table B-2

Computing System Control Categories

- I Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazard's occurrence.
- IIa Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.
- IIb Software item displays information requiring immediate operator action to mitigate a hazard. Software failures may allow or fail to prevent the hazard's occurrence.
- IIIa Software item issues commands over potentially hazardous hardware systems, subsystems or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.
- IIIb Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.
- IV Software does not control safety critical hardware systems, subsystems or components and does not provide safety critical information.

Table B-3
Computing System Control Category
Assignment Examples

	Computing System Control Categories			
	I	II	III	IV
Fuzes				
Intelligent fuzes, separate mechanical Safe and Arm (S&A) devices or units		X		
Intelligent fuzes, computer controlled S&A	X			
Electronic S&A devices with computer control	X			
Mechanical fuzes				X
Seekers				
Intelligent seekers	X			
Intelligent seekers with Independent Identification Friend or Foe (IFF)		X		
Guidance Systems				
Computer controlled with no recall or abort	X			
Commanded self-destruct capability		X		
Operator commanded systems			X	
With map display feedback	X			

Annex B to
STANAG 4404
(Edition 1)

Table B-3 (Continued)
Computing System Control Category
Assignment Examples

Computing System Control Categories

	I	II	III	IV
Target Allocation Systems				
With independent cross-check			X	
Without independent cross-check	X			
Launcher Control Systems				
Computer controlled	X			
Computer controlled with independent safety features		X		
Pointing and Firing Cutout Systems				
Computer controlled	X			
Computer controlled with mechanical backup		X		
Command and Control (C&C) Systems				
General C&C functions with external safety features		X		
Autonomous C&C systems	X			

Table B-4
Design Requirement and Guideline Tailoring
Program Development Phase Applicability

Design Guideline Reference	C	PD	DD	Code	Test	Integ	Mods
6. Design and development process							
6.1 Configuration control	R	V	V	V	V	V	V
6.2 Quality Assurance Program	R	R	R	R	R	R	R
6.3 Two person rule	R	R	R	R	N	N	R
6.4 Reviews and audits	N	R	R	V	N	N	R/V
6.5 Program patch prohibition	R	N	N	N	R/T	V/T	V
6.6 Design Verification/Validation	R	R	R	R	R	R	R
7. System design							
7.1 Designed safe states.	R	R	V	V	T	T	R/V
7.2 Standalone computer	O	O	O	O	O	O	O
7.3 Ease of maintenance	O	R	R	V	T	T	R/V
7.4 Safe state return	R	R	V	V	V/T	V/T	R
7.5 Restoration of interlocks	R	R	V	V	T	T	R/V
7.6 Input/Output registers	R	R	V	V	T	T	R/V
7.7 External hardware failures	R	R	V	V	T	T	R/V
7.8 Safety kernel failure	R	R	V	V	T	T	V
7.9 Circumvent unsafe conditions	O	R	R	V	T	T	R/V
7.10 Fallback and recovery	R	R	V	V	T	T	R/V
7.11 Simulators	R	R	R	V	T	T	R/V
7.12 System error log	R	R	V	V	T	T	V
7.13 Positive feedback mechanisms	R	R	V	V	T	T	R/V
7.14 Peak load conditions	N	R	R	V	T	T	R/V

Table B-4 (Continued)
Design Requirement and Guideline Tailoring
Program Development Phase Applicability

Design Guideline Reference	C	PD	DD	Code	Test	Integ	Mod
8. Power-up Initialization Requirements							
8.1 Power Up initialization	R	R	V	V	T	T	R/V
8.2 Power faults	R	R	V	V	T	T	V
8.3 Primary computer failure	R	R	V	V	T	T	R/V
8.4 Maintenance interlocks	R	R	R	V	T	T	V
8.5 System level check	R	R	V	V	T	T	V
9. Computing System Hardware Requirements							
9.1 CPU selection	R	R	N	N	N	N	N
9.2 Minimum clock cycles	N	R	R	V	T	T	T
9.6 Read only memories	N	O	R	R	T	T	T
10. Self-Check Design Requirements							
10.1 Watchdog timers	R	R	V	V	T	T	T
10.2 Memory checks	R	R	V	V	T	T	R/V
10.3 Fault detection	R	R	V	V	T	T	R/V
10.4 Operational checks	R	R	V	V	T	T	R/V
11. Safety Critical Computing System Functions Protection Requirements							
11.1 Safety degradation	R	R	R	V	T	T	R/V
11.2 Unauthorized interaction	R	R	R	V	T	T	R/V
11.3 Unauthorized access	R	R	V	V	T	T	R/V
11.4 Safety kernels ROM	R	R	R	V	N	N	R/V
11.5 Safety kernel independence	R	R	V	V	T	T	R/V
11.6 Inadvertent jumps	R	R	V	V	T	T	R/V
11.7 Load data integrity	R	R	R	R	V	V	R/V
11.8 Operational reconfiguration integrity	R	R	R	R	V	V	R/V

Table B-4 (Continued)
Design Requirement and Guideline Tailoring
Program Development Phase Applicability

Design Guideline Reference	C	PD	DD	Code	Test	Integ	Mod
12. Interface Design Requirements							
12.1 Feedback loops	N	R	V	V	T	T	R/V
12.2 Interface control	R	R	V	V	T	T	R/V
12.3 Decision statements	R	R	V	V	T	T	R/V
12.4 Inter-CPU communications	R	R	V	V	T	T	R/V
12.5 Data transfer messages	R	R	V	V	T	T	R/V
12.6 External functions		R	R	V	V	T	T
R/V							
12.7 Input reasonableness checks	R	R	V	V	T	T	R/V
12.8 Full scale representations	R	R	V	V	T	T	R/V
13. User Interface Design							
13.1 Processing cancellation	R	R	V	V	T	T	R/V
13.2 Hazardous function initiation	R	R	V	V	T	T	R/V
13.3 Safety critical displays	R	R	V	V	T	T	R/V
13.4 Operator entry errors	R	R	V	V	T	T	R/V
13.5 Safety critical alerts	R	R	V	V	T	T	R/V
13.6 Unsafe situation alerts	R	R	V	V	T	T	R/V
13.7 Unsafe state alerts		R	R	V	V	T	T
R/V							
14. Software Design and Coding Requirements							
14.1 Modular code	R	R	V	V	N	N	R/V
14.2 Number of modules	O	R	R	V	V	N	R/V
14.3 Execution path	R	R	V	V	T	T	R/V
14.4 Halt instructions	N	R	R	R	V	N	R/V
14.5 Single purpose files	N	R	R	V	V	N	R/V
14.6 Unnecessary features	O	R	V	V	T	T	R/V
14.7 Indirect addressing methods	R	R	V	V	T	T	R/V

Table B-4 (Continued)
Design Requirement and Guideline Tailoring
Program Development Phase Applicability

Design Guideline Reference	C	PD	DD	Code	Test	Integ	Mod
14. Software Design (continued)							
14.8 Uninterruptable code	R	R	O	O	T	T	R/V
14.9 Safety critical files	R	R	R	V	T	N	R/V
14.10 Unused memory	R	R	V	V	T	T	R/V
14.11 Overlays	R	R	V	V	T	T	R/V
14.12 Operating system functions	O	R	V	V	N	N	R/V
15. Software Coding Requirements							
15.1 Compiler	N	N	N	R	T	N	N
15.2 Flags and variables	N	R	R	V	V	N	R/V
15.3 Loop entry point	O	R	R	V	T	N	R/V
15.4 Software maintenance design	R	R	R	V	T	T	R/V
15.5 Variable declaration	N	R	R	V	T	T	R/V
15.6 Unused executable code	R	R	R	R	V	N	R/V
15.7 Unreferenced variables	R	R	R	V	V	T	R/V
15.8 Assignment statements	O	R	R	V	V	N	R/V
15.9 Conditional statements	O	R	R	V	T	T	R/V
15.10 Strong data typing	R	R	V	V	T	T	R/V
15.11 Timer values annotated	R	R	R	R	N	N	R
15.12 Critical variable identification	R	R	R	R	R	R	R
15.13 Global Variable	N	R	R	V	T	N	R

Table B-4 (Continued)
Design Requirement and Guideline Tailoring
Program Development Phase Applicability

Design Guideline Reference	C	PD	DD	Code	Test	Integ	Mod
16. Software Maintenance Requirements							
16.1 Critical function changes	R	N	N	N	N	N	R
16.2 Critical firmware changes	R	N	N	N	N	N	R
16.3 Software change medium	R	N	N	N	N	N	R
16.4 Modification configuration control	N	R	R	R	V	V	R
16.5 Version identification	R	R	R	R	R	R	R
17. Software Analysis and Testing							
17.1 Formal test coverage	R	R	R	R	V	V	R
17.2 GO/NO-GO path testing	N	R	R	R	T	T	R
17.3 Input failure modes	R	R	R	R	T	T	R/V
17.4 Boundary test conditions	N	R	R	R	T	T	R
17.5 Input data rates	N	N	N	N	T	T	N
17.6 Zero value testing	N	R	R	R	T	T	R
17.7 Regression testing		R	R	R	R	T	T
R/V							
17.8 Operator interface testing	R	R	R	R	T	T	R/V
17.9 Duration stress testing	N	N	N	N	R/T	R/T	R